FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

COMPUTER APPLACATION LAB

ENCS4110

Report#1

EXP.No.3.ARM'S Flow Control Instructions

.....

Student Name: saja asfour

Student ID: 1210737

Instructor name: Abdel Salam Sayyad

Section number:2

Date: 22/4/2023

3.1 Abstract:

-The Aim of the experiment: to understand the instruction of ARM , and understand the branch instruction and how used strings.

-Equipment Used in the experiment: keil vision 5.

Contents:

List of figure:

3.2 Theory

3.2.1 Introduction:

*ARM is processor based in microcontroller , ARM architecture developed by using RISC Machine .

*All instruction are 32-bit fixed length in general but in Thumb the long of it equal 16 bit .

*In ARM have 16 registers(R0-R15).

*The registers can be divided into:
- R0-R7 low and general purpose registers
-R8-R12 High and general purpose registers
-R13 stack pointer (SP)
-R14 link register(LR)
-R15 program Counter (PC)

*In addition there are a special registers for example current program statuses register (CPSR) 32-bit this register is consist of condition flags:
-N: negative or less than flag
-Z : zero flags
-C: Carry or borrow or extended flag
-V over flow flag
Also, in CPSR it contains the control bit to control the system at least significant 8 bit.

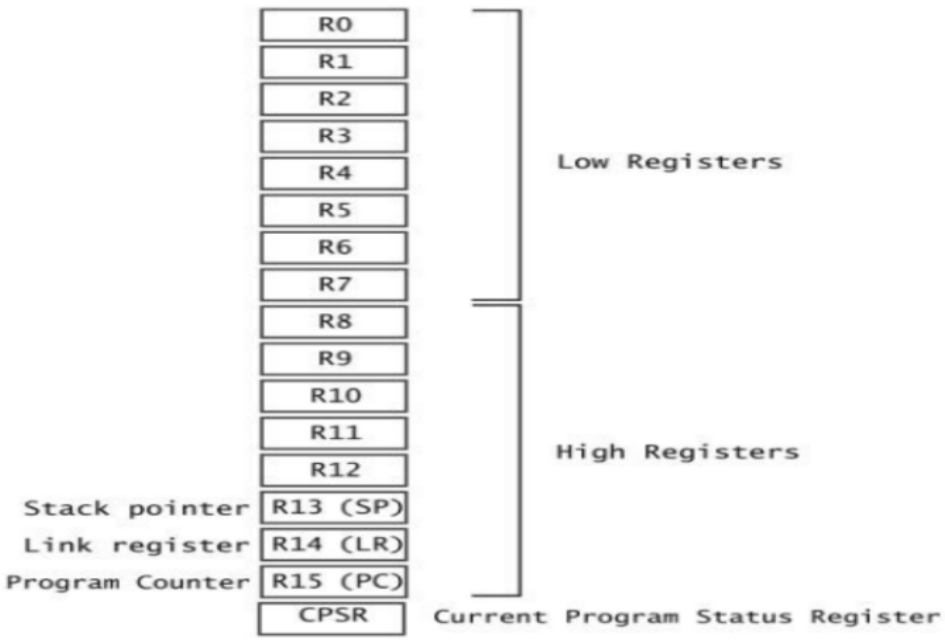

Figure 1: ARM registers

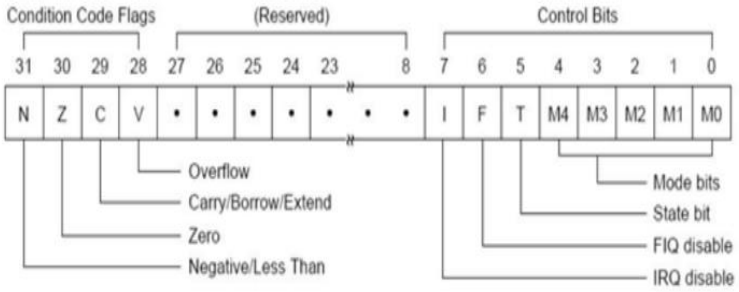

Figure 2: CPSR

*The condition code flag change in some arithmetic and logic instructions for example:
SUBS R1,R2,R3 → R1=R2-R3 flags are update
Another example in logic:
ANDS R1,R2,R3→ R1=R2 AND R3 also update the flag
But when write without 'S' performs operations but the flags not change.

*In branch instructions depends on condition flag if the value in flag match the conditions for the processor conditions occur, they will executed the branch .

3.2.2 Branch and control instruction:

*Used the branch instruction in loop and selection.
*Some branch and control instruction:
-Unconditional branch :
- B Loop A → branch to label A

```
LoopA
        ADD     R1,R1,#1
        SUB     R2,#1
        B       LoopA
```

Figure 3: B Branch

After increment R1 and decrement R2 Branch to LoopA(The same Loop)
-Some conditional Branch:
- BEQ : branch when Z=1
- BNE : branch when Z=0
- BMI : branch when N=1
- BPL : branch when N=0

```
LoopA
        ADD     R1,# 1
        SUB     R2,#1
        BEQ     DONE
        B       LoopA
DONE
```

Figure 4: BEQ Branch

If the value in R2 equal zero→ if Z=1 so will branch to label DONE

-BLT: save the next address link –PC- in link register(LR) , the instruction execute when:

-N flag is clear (equal 0) , V flag is set (equal 1)

- N flag is set , V flag is clear

This mean : less than→ (N EOR V)=1

-BLE: save PC in LR , the condition are met :

-Z flag is set

-N flag is clear, V is set

-N flag is set , V is clear

This mean: less than or equal→(Z ORR(N EOR V))=0

-BGT: save PC in LR , the condition to execute:

-Z,N and V flags are all clear

-Z flag is clear , N and V flags are set

This mean: greater than→(Z ORR(N EOR V))=0

-BGE: save PC in LR , the condition are:

-Z flag is set ,N,V flags are clear

-Z, N and V flags are set

This mean: greater than or equal→(N EOR V)=0

-BL: stored address in Link register (LR) before branch , then restore the PC to LR to Complete the instruction. LR must have stacked to store the maximum number address

-BX LR : branch and store the address in register "branch indirect"

-BXL R0: branch with link , and optionally exchange instruction set and store the Address in R0.

3.2.3 Compere instructions:

-CBZ R1,LABEL→ if R1 equal 0 then Z=1 so branch to LABEL

-CBNZ R1, LABEL1→ if R1 is not equal 0 then Z=0 so branch to LABEL1

-CMP R1,R2→ compare R1-R2 , then update the flags

-CMN R1,R2→compare negative R1-R2(R1+R2) then update the flags

-CMPGT SP,R7,LSL#2→ shift left 2-bit for R7 then compare with SP then update the flags

3.3 Procedure

3.3.1 Example using Branch instructions:

In this example we will count the length of String.

3.3.1.1 Code:

```
1   PRESERVE8
2   THUMB
3
4
5    AREA    RESET, DATA, READONLY
6    EXPORT  __Vectors
7   __Vectors
8   DCD  0x20001000
9     DCD  Reset_Handler
10    ALIGN
11
12  string1
13      DCB "Hello world!",0
14  SUM DCD 0
15
16   AREA    MYCODE, CODE, READONLY
17   ENTRY
18   EXPORT Reset_Handler
19  Reset_Handler
20
21
22      LDR R0, = string1 ; Load the address of string1 into the register R0
23      MOV R1, #0 ; Initialize the counter counting the length of string1
24  loopCount
25      LDRB R2, [R0] ; Load the character from the address R0 contains
26      CMP R2, #0
27      BEQ countDone
28  ; If it is zero...remember null terminated...
29  ; You are done with the string. The length is in R1.
30      ADD R0, #1 ; Otherwise, increment index to the next character
31      ADD R1, #1 ; increment the counter for length
32      B loopCount
33  countDone
34  STOP
35      B STOP
36      END
```

Figure 5: code for example 1

3.3.1.2 Simulation and discuassion:

-I used DCB to declare an initalized byte (8 bit) for memory variables.

- 0(null) , in the end of string to know that the last string has reached.

- in line 22 I used LDR to load the address of string1 to R0 , to use this address to load the char from memory to register then can make any operation of it.

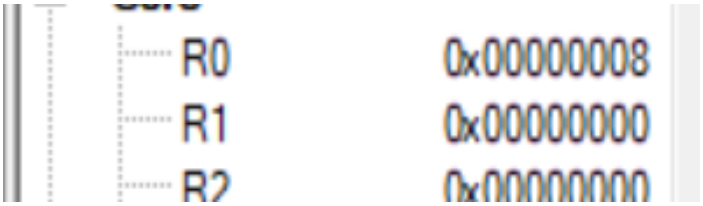

Figure 6: register window for R0 ,R1 in example 1

-In R0 =0x00000008 this address in memory allocated the data (string 1)

-In R1 =0x00000000

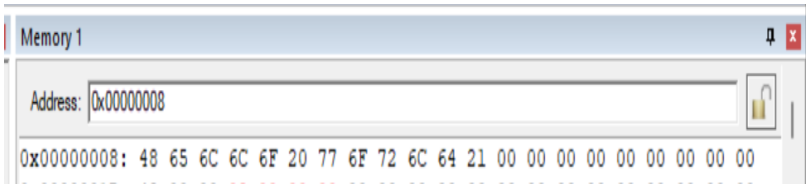

Figure 7: memory window for R1 address in example1

Address 0x00000008 stored it the data ; '48'→ ASCII code for 'H'

; '65'→ ASCII code for 'e'

; and so on...

-In line 24 : LoopCount (liabel): this loop keep repeating in case the char is not equal zero,and it doesn't reach the end of the string.

-In line 25: 'LDRB' load byte(8 bit) from address memory in R0 to R2

-Then compare the value in R2 with 0 →R2-0 ; if the result equal 0 this mean R2=0 so reach the end of the string and will stop and finish program . when the result from 'CMP' equal 0 then Z flag=1,so can used 'BEQ' branch condition by branch to liable countDone .Else if R2!=0 after compare this mean we aren't reach the end of the string and incrament for address to next char and increament the length of the string then branch for begin Loop used B. the length of string is stored in R1.

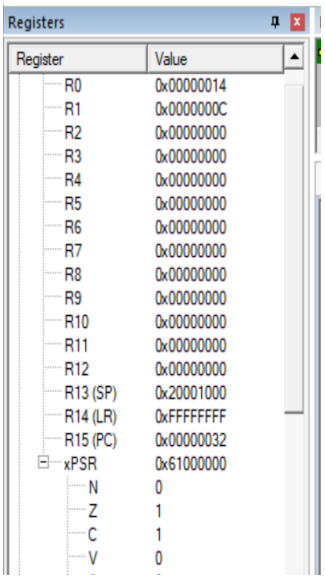

Figure 8: the result of register in example1

3.3.2 Another example:

This example to clear branch instruction

3.3.2.1 Code:

exp3.s

```
4
5               AREA    RESET, DATA, READONLY
6               EXPORT  __Vectors
7   __Vectors
8             DCD  0x20001000
9                DCD  Reset_Handler
10               ALIGN
11
12  SUMP DCD SUM
13  N DCD 5
14          AREA MYRAM, DATA, READWRITE
15  SUM DCD 0
16
17              AREA    MYCODE, CODE, READONLY
18              ENTRY
19              EXPORT Reset_Handler
20  Reset_Handler
21
22     LDR R1, N ;Load count into R1
23     MOV R0, #0 ;Clear accumulator R0
24  LOOP
25      ADD R0, R0, R1 ;Add number into R0
26
27      SUBS R1, R1, #1 ;Decrement loop counter R1
28      BGT LOOP ;Branch back if not done
29      LDR R3, SUMP ;Load address of SUM to R3
30      STR R0, [R3] ;Store SUM
31      LDR R4, [R3]
32  STOP
33       B STOP
34       END
```

Figure 9: code for example 2

3.3.2.2 simulation and discussion:

-In line 12: 'DCD' declare an initialized word (32-bit) for memory variable. SUMP is pointer of SUM, when SUM initialized in memory equal 0.

- In line 22: load the value in R1(count)

-In line 23: 'MOV' to clear the R0 to used it for ADD operation

- In line 24: this loop keeps if R1!=0, when R1 =0 end the program

- In line 25: 'ADD' sum R0 with number in R1

- In line 27: decrement the counter(R1) and update the flags

If Z=0, V=N this mean counter not equal 0 so branch for loop liable .But if Z=1 the Counter equal 0 so not keep in loop, so complete the code .

- In line 29: 'LDR' load the address of SUMP in R3

-In line 30: store the result of sum from R0 to address memory [R3]

-In line 31: load the sum from memory to R4

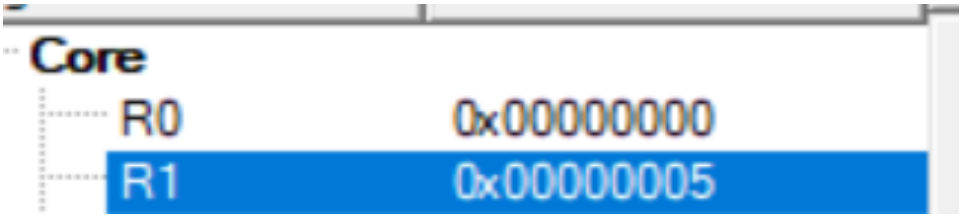

Figure 10: line 22 and 23 register window for example 2

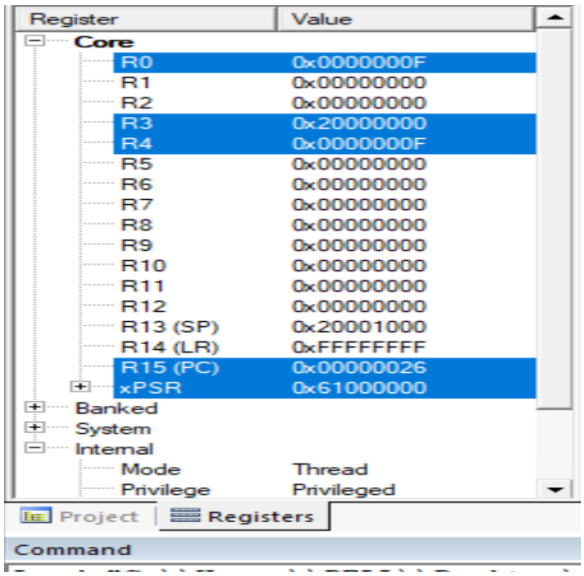

Figure 11: the register window for example 2

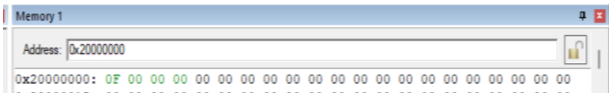

Figure 12: memory window for example 2

-The address of memory which has been stored the sum in it

-NOTE: the memory is big endian → the least significant byte in register stored in most significant byte in memory.

3.4 Lab Work

3.4.1 Question:

- Write an ARM assembly program CountVowelsOne.s to count how many vowels and Non-vowels in the following string:

"ARM assembly language is important to learn!",0

3.4.2 Code:

```
        PRESERVE8
        THUMB
         AREA    RESET, DATA, READONLY
         EXPORT  __Vectors
__Vectors
       DCD  0x20001000
          DCD  Reset_Handler
          ALIGN
string1
      DCB "ARM assembly language is important to learn!",0 ; store the char in memory one byte 8-bit
string2
      DCB 'a','o','u','i','e',0 ;
        AREA    MYCODE, CODE, READONLY
        ENTRY
        EXPORT Reset_Handler
Reset_Handler
     LDR R0, = string1 ; Load the address of string1 into the register R0
     MOV R1, #0 ;counter for vowels
     MOV R2, #0 ; counter for NON-vowels
     LDR R4, = string2; load sddress of char vowels into R4
loopCount
     LDRB R5, [R0] ; Load the character(byte 8-bit) from the address contains in R0 to R5
     CMP R5, #0    ; compare the char in R5 with zero if equal zero end the string
     BEQ countDone  ;if Z=1 -->end program
     ORR R5,R5,0x00000020 ; to convert the char from captial leter to small
loop
     LDRB R6 ,[R4]   ;load to vowel char
     CMP R6,#0  ;R6 equal zero -->end of string2 charecters of vowels
     BEQ nonvowels  ; branch of nonvowel
     CMP R6,R5 ;R6-R5   ;copmere
     BEQ vowels   ;branch of vowels
     ADD R4,R4,#1 ;incremt address of vowels next index
     B loop
vowels
     ADD R1,R1,#1 ; counter vowel +1
```

```
vowels
     ADD R1,R1,#1 ; counter vowel +1
     ADD R0,R0,#1 ;increment the address  get the next char
     LDR R4, = string2; reset R4
     B loopCount
nonvowels
        ADD R2,R2,#1
        ADD R0,R0,#1
        LDR R4, = string2;reset R4
        B loopCount
countDone
STOP
    B STOP
    END
```

Figure 13: code for lab work

3.4.3 Simulation and discussion:

-In line 10: used 'DCB' to declare an initialized byte(8 bit) for memory , so each char Stored in one byte in memory .'0' in the end of string to know that we are in the end of string.

- used 'LDR' to load the address of string1 into R0 and load address of String2 into R4

-used 'MOV' to initialized the counter counting of char vowels and non-vowels

-load the char from the memory used indirect address → 'LDRB' used to load the char from The address R0 contains then, compare the char with zero if it equal zero then we reach the End of the string then end the program else, complete the code and check if char vowel or Not.

- used 'ORR' to convert the char from capital to small.

-From line 26: 'LDRB' load the character from address R4 contains. Compare the value on R0 with zero if result equal 0 then Z flag =1 then we reach the end of string2 (vowel char) this mean char is non-vowel due to comparing all char in string 2 with the char in R5 so branch to nonvowels labile .

-In nonvowels: increment each the counter of nonvowels char and index of the next char want to check it. Then brach to the loop counter.

-if the result of compare is not equal 0, then z flag=0, then compare between two register R6-R5 is equal 0 conclusion the char char is vowel so branch the vowel labile

-in vowels : increment each the count of vowels char and index of the next char want to check it and then branch to loopcount

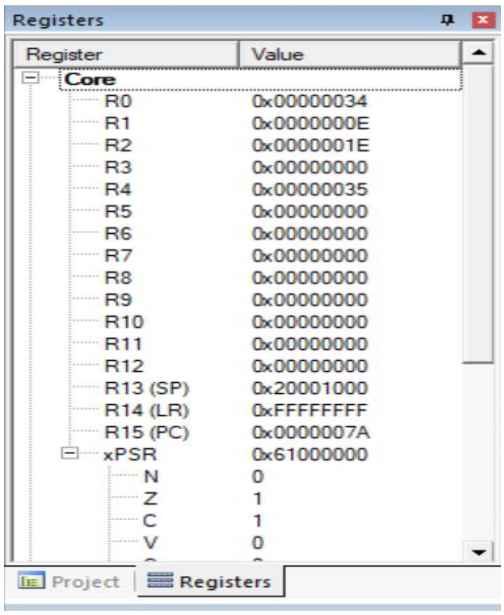

Figure 14: the register window for lab work

-The number of char vowel is →(0x0000000E) =14

-The number of char non-vowel is→ (0x0000001E)=30

-R5,R6=0X00000000 due to reach the end of string and Z flag=1

3.5 Conclusion

In this experiment I understood the control and branch instruction , how used it and when can used such as in loop. There are some branch need condition on flags and other don't need . the use of string was also known. In addition , understood the compare instruction , where it update the flags. And in the final of our lab we make "To Do" from the assistant.

3.6 References

- https://developer.arm.com/documentation/den0013/d/Introduction-to-Assembly-Language#:~:text=Assembly%20language%20is%20a%20low,to%20code%20in%20assembly%20language
- https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/The-Instruction-Sets/Branch-instructions